Straight-line instruction sequence completeness for total calculation on cancellation meadows*

Jan A. Bergstra
Inge Bethke

Section Software Engineering, Informatics Institute, University of Amsterdam
URL: `www.science.uva.nl/~{inge,janb}`

Abstract

A combination of program algebra with the theory of meadows is designed leading to a theory of computation in algebraic structures which use in addition to a zero test and copying instructions the instruction set $\{x \Leftarrow 0, x \Leftarrow 1, x \Leftarrow -x, x \Leftarrow x^{-1}, x \Leftarrow x + y, x \Leftarrow x \cdot y\}$. It is proven that total functions on cancellation meadows can be computed by straight-line programs using at most 5 auxiliary variables. A similar result is obtained for signed meadows.

Key words: Program algebra, Instruction sequences, Execution of programs, Straight-line programs, Division-by-zero, Fields, Meadows, Equational specification, Calculation in meadows.

1 Introduction

Program algebra is an approach to the formal description of the semantics of programming languages. It is a framework that permits algebraic reasoning about programs and has been investigated in various settings (see e.g. [4, 12, 13, 14, 18]).

The theory of fields is a very active area which is not only of great theoretical interest but has also found applications both within mathematics—combinatorics and algorithm analysis—as well as in engineering sciences and, in particular, in coding theory and sequence design. Unfortunately, since fields are not axiomatized by equations only, Birkhoff's Theorem fails, i.e. fields do not constitute a variety: they are not closed under products, subalgebras, and homomorphic images. In [9], the concept of *meadows* was introduced, structures very similar to fields—the considerable difference being that meadows enjoy a total multiplicative inversion and do form a variety.

*Partially supported by the Dutch NWO project *Thread Algebra for Strategic Interleaving*, project number 612.000.421.

The aim of this paper is to combine these two areas of research in order to create a theory of computation in algebraic structures which can be used to investigate questions of definability and complexity.

Many computations in applied mathematics can be formulated as computations on fields. In many cases such computations terminate on all inputs yielding total functions. Replacing fields by meadows, which simplify their equational logic, we investigate properties of instruction sequences which compute total functions on all meadows. We shall prove that total functions on cancellation meadows—meadows which in addition satisfy the inverse law known form the theory of fields—can be computed by *straight-line programs* with a bound supply of auxiliary variables. These kind of programs have been amply investigated and simplification and equivalence problems for several classes of straight-line programs over varying instruction sets are known (see e.g. [15, 16]).

The paper is organized as follows. In the next section we recall the basics of program algebra, thread algebra and meadows. Here the notion of program algebra refers to the concept introduced in [4] which focuses on instruction sequences. In Section 3 we introduce instruction sequences for functions on the rational numbers. The main theorem is proven in Section 4. We prove that total functions on cancellation meadows can be represented by a normal form without tests and jumps which uses at most 5 auxiliary variables. This result is extended to signed cancellation meadows—cancellation meadows that presuppose an ordering of its domain—in Section 5.

2 The basics of program algebra, thread algebra and meadows

In this section we recall program algebra, thread algebra and meadows.

2.1 Program algebra

The programm algebra PGA was introduced in [4].

Assume A is a set of constants with typical elements $\mathtt{a}, \mathtt{b}, \mathtt{c}, \ldots$. Instruction sequences are of the following form ($k \in \mathbb{N}$):

$$I ::= \mathtt{a} \mid +\mathtt{a} \mid -\mathtt{a} \mid \#k \mid ! \mid I;I \mid I^{\omega}.$$

The first five forms above are called *primitive instructions.* These are

- *basic instructions* a which prescribe behaviours that are considered indivisible and executable in finite time, and which return upon execution a Boolean reply value,
- *test instructions* obtained from basic instructions by prefixing them with either a + (positive test instruction) or a − (negative test instruction) which control subsequent execution via the reply of their execution,

$$(X;Y);Z = X;(Y;Z) \tag{PGA1}$$

$$(X^n)^\omega = X^\omega \tag{PGA2}$$

$$X^\omega;Y = X^\omega \tag{PGA3}$$

$$(X;Y)^\omega = X;(Y;X)^\omega \tag{PGA4}$$

$$\#n{+}1;u_1;\ldots;u_n;\#0 = \#0;u_1;\ldots;u_n;\#0 \tag{PGA5}$$

$$\#n{+}1;u_1;\ldots;u_n;\#m = \#n{+}m{+}1;u_1;\ldots;u_n;\#m \tag{PGA6}$$

$$(\#k{+}n{+}1;u_1;\ldots;u_n)^\omega = (\#k;u_1;\ldots;u_n)^\omega \tag{PGA7}$$

$$X = u_1;\ldots;u_n;(v_1;\ldots;v_{m+1})^\omega \rightarrow\ \#n{+}m{+}k{+}2;X = \#n{+}k{+}1;X \tag{PGA8}$$

Table 1: PGA-axioms for single-pass congruence

- *jump instructions* $\#k$ which prescribe to jump k instructions ahead—if possible; otherwise deadlock occurs—and generate no observable behavior, and
- the *termination instruction* ! which prescribes successful termination, an event that is taken to be observable.

Finite instruction sequences are obtained from primitive instructions using *concatenation*: if I and J are finite instruction sequences, then so is

$$I;J$$

which is the instruction sequence that lists J's primitive instructions right after those of I. A special subclass of the finite instruction sequences are the so-called *straight-line instruction sequences* which are finite instruction sequences *without* tests and jumps.

Periodic instruction sequences are defined using the repetition operator: if I is an instruction sequence, then

$$I^\omega$$

is the instruction sequence that repeats I forever, thus $I;I;I;\ldots$.

In PGA, different types of equality are discerned, the most simple of which is *single-pass congruence*, identifying sequences that execute identical instructions. For finite instruction sequences, single-pass congruence boils down to the associativity of concatenation, and is axiomatized by

$$(X;Y);Z = X;(Y;Z).$$

In the sequel we leave out brackets in repeated concatenations. In the case of infinite instruction sequences, additional axioms are needed. Define $X^1 = X$ and for $n > 0$, $X^{n+1} = X;X^n$. According to [4], single-pass congruence for arbitrary instruction sequences is axiomatized by the axiom schemes PGA1-PGA4 in Table 1.

Using the axioms PGA1–PGA4 and thus preserving single-pass congruence, each instruction sequence can be rewritten into one of the following forms:

Y not containing repetition, or

$Y; Z^\omega$ with Y and Z not containing repetition.

Any instruction sequence in one of the two above forms is said to be in *first canonical form*.

Instruction sequences in first canonical form can be converted into *second canonical form*: a first canonical form in which no chained jumps occur, i.e., jumps to jump instructions (apart from #0), and in which each non-chaining jump into the repeating part is minimized. The associated congruence is called *structural congruence* and is axiomatized in Table 1. Note that axiom PGA8 is an equational axiom, the implication is only used to enhance readability.

Two examples, of which the right-hand sides are in second canonical form:

$$\#2; \mathtt{a}; (\#5; \mathtt{b}; +\mathtt{c})^\omega =_{sc} \#4; \mathtt{a}; (\#2; \mathtt{b}; +\mathtt{c})^\omega,$$
$$+\mathtt{a}; \#2; (+\mathtt{b}; \#2; -\mathtt{c}; \#2)^\omega =_{sc} +\mathtt{a}; \#0; (+\mathtt{b}; \#0; -\mathtt{c}; \#0)^\omega.$$

For each instruction sequence there exists a structurally equivalent second canonical form.

For more information on PGA we refer to [4, 17].

2.2 Thread algebra

Thread algebra is the behavioural semantics for PGA and was introduced in e.g. [1, 4] under the name Polarized Process Algebra.

Finite threads are defined inductively by:

S – *stop*, the termination thread,

D – *inaction* or *deadlock*, the inactive thread,

$T \trianglelefteq \mathtt{a} \trianglerighteq T'$ – the *postconditional composition* of T and T' for action $\mathtt{a}$, where T and T' are finite threads and $\mathtt{a} \in A$.

The behavior of the thread $T \trianglelefteq \mathtt{a} \trianglerighteq T'$ starts with the *action* $\mathtt{a}$ and continues as T upon reply `true` to $\mathtt{a}$, and as T' upon reply `false`. Note that finite threads always end in S or D. We use *action prefix* $\mathtt{a} \circ T$ as an abbreviation for $T \trianglelefteq \mathtt{a} \trianglerighteq T$ and take $\circ$ to bind strongest.

For every finite thread there exists a finite upper bound to the number of consecutive actions it can perform. The *approximation operator* π_n gives the behaviour up to depth n and is defined by

1. $\pi_0(T) = \mathsf{D}$,
2. $\pi_{n+1}(\mathsf{S}) = \mathsf{S}$,
3. $\pi_{n+1}(\mathsf{D}) = \mathsf{D}$, and
4. $\pi_{n+1}(T \trianglelefteq \mathtt{a} \trianglerighteq T') = \pi_n(T) \trianglelefteq \mathtt{a} \trianglerighteq \pi_n(T')$

$$
\begin{aligned}
|!| &= \mathsf{S} & \qquad |!;X| &= \mathsf{S}\\
|\mathsf{a}| &= \mathtt{a} \circ \mathsf{D} & |\mathsf{a};X| &= \mathtt{a} \circ |X|\\
|{+}\mathsf{a}| &= \mathtt{a} \circ \mathsf{D} & |{+}\mathsf{a};X| &= |X| \trianglelefteq \mathtt{a} \trianglerighteq |\#2;X|\\
|{-}\mathsf{a}| &= \mathtt{a} \circ \mathsf{D}, & |{-}\mathsf{a};X| &= |\#2;X| \trianglelefteq \mathtt{a} \trianglerighteq |X|\\
\\
|\#k| &= \mathsf{D} & |\#0;X| &= \mathsf{D}\\
& & |\#1;X| &= |X|\\
& & |\#k{+}2;u| &= \mathsf{D}\\
& & |\#k{+}2;u;X| &= |\#k{+}1;X|
\end{aligned}
$$

Table 2: Equations for thread extraction, where $\mathtt{a}$ ranges over the basic instructions, and u over the primitive instructions ($k \in \mathbb{N}$)

for finite threads T, T' and $n \in \mathbb{N}$. Infinite threads are obtained as *projective sequences* of finite threads of the form $(T_n)_{n \in \mathbb{N}}$ where for every $n \in \mathbb{N}$, $\pi_n(T_{n+1}) = T_n$.

Upon its execution, a basic or test instruction yields the equally named action in a post conditional composition. Thread extraction on PGA, notation

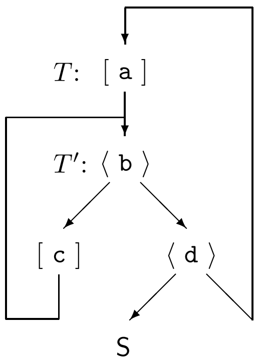
$$|X|$$

with X an instruction sequence, is defined by the thirteen equations in Table 2. In particular, note that upon the execution of a positive test instruction $+\mathsf{a}$, the reply `true` to `a` prescribes to continue with the next instruction and `false` to skip the next instruction and to continue with the instruction thereafter; if no such instruction is available, deadlock occurs. For the execution of a negative test instruction $-\mathsf{a}$, subsequent execution is prescribed by the complementary replies.

For an instruction sequence in second canonical form, these equations either yield a finite thread, or a so-called *regular* thread, i.e., a finite state thread in which infinite paths can occur. Each regular thread can be specified (defined) by a finite number of recursive equations. As an example, the regular thread T specified by

$$
\begin{aligned}
T &= \mathtt{a} \circ T'\\
T' &= \mathtt{c} \circ T' \trianglelefteq \mathtt{b} \trianglerighteq (\mathsf{S} \trianglelefteq \mathtt{d} \trianglerighteq T)
\end{aligned}
$$

can be defined by $|\mathsf{a};({+}\mathsf{b};\#2;\#3;\mathsf{c};\#4;{+}\mathsf{d};!;\mathsf{a})^\omega|$. A picture of this thread is



This thread can also given by the projective sequence $(\pi_n(T))_{n\in\mathbb{N}}$ where

$$\begin{array}{rcl}
\pi_0(T) & = & \mathsf{D} \\
\pi_1(T) & = & \mathtt{a} \circ \mathsf{D} \\
\pi_2(T) & = & \mathtt{a} \circ b \circ \mathsf{D} \\
\pi_3(T) & = & \mathtt{a} \circ (\mathtt{c} \circ \mathsf{D} \trianglelefteq \mathtt{b} \trianglerighteq \mathtt{d} \circ \mathsf{D})
\end{array}$$

and $\pi_{n+4}(T) = \mathtt{a} \circ (\mathtt{c} \circ \pi_{n+1}(T') \trianglelefteq \mathtt{b} \trianglerighteq (\mathsf{S} \trianglelefteq \mathtt{b} \trianglerighteq \pi_{n+1}(T)))$. Observe that thread extraction of straight-line instruction sequences yield finite and test-free threads.

For basic information on thread algebra we refer to [2, 17]; more advanced matters, such as an operational semantics for thread algebra, are discussed in [5]. We here only mention the fact that each regular thread can be specified in PGA, and, conversely, that each PGA-program defines a regular thread.

2.3 Meadows

A meadow [3, 9] is a commutative ring with unit equipped with a total unary operation $(\_)^{-1}$ named *inverse* that satisfies the two equations

$$\begin{array}{c}
(x^{-1})^{-1} = x, \\
x \cdot (x \cdot x^{-1}) = x. \quad (\mathit{RIL})
\end{array}$$

Here *RIL* abbreviates *Restricted Inverse Law.* We write *Md* for the set of axioms in Table 3.

In the meadow $\mathbb{Q}$ of rational numbers, every element has a restricted inverse. If $x \neq 0$, the inverse is just the "regular" inverse, and $0^{-1} = 0$. Another example is ring $\mathbb{Z}/6\mathbb{Z}$ with elements $\{0, 1, 2, \ldots, 5\}$ where arithmetic is performed modulo 6. We find that every element has a restricted inverse as follows:

$$\begin{array}{rclcrcl}
(0)^{-1} & = & 0 & \qquad & (1)^{-1} & = & 1 \\
(2)^{-1} & = & 2 & & (3)^{-1} & = & 3 \\
(4)^{-1} & = & 4 & & (5)^{-1} & = & 5
\end{array}$$

A characterization of finite meadows can be found in [11]. From the axioms in *Md* the

$$\begin{aligned}
(x+y)+z &= x+(y+z)\\
x+y &= y+x\\
x+0 &= x\\
x+(-x) &= 0\\
(x\cdot y)\cdot z &= x\cdot(y\cdot z)\\
x\cdot y &= y\cdot x\\
1\cdot x &= x\\
x\cdot(y+z) &= x\cdot y + x\cdot z\\
(x^{-1})^{-1} &= x\\
x\cdot(x\cdot x^{-1}) &= x
\end{aligned}$$

Table 3: The set *Md* of axioms for meadows

following identities are derivable:

$$\begin{aligned}
(0)^{-1} &= 0,\\
(-x)^{-1} &= -(x^{-1}),\\
(x\cdot y)^{-1} &= x^{-1}\cdot y^{-1},\\
0\cdot x &= 0,\\
x\cdot -y &= -(x\cdot y),\\
-(-x) &= x.
\end{aligned}$$

We write $\Sigma_m = (0,1,+,\cdot,-,{}^{-1})$ for the signature of meadows and $\mathit{Ter}(\Sigma_m,X)$ for the set of open meadow terms with free variables in X. For $t,u\in \mathit{Ter}(\Sigma_m,X)$ we shall often write $1/t$ or

$$\frac{1}{t}$$

for t^{-1}, tu for $t\cdot u$, t/u for $t\cdot 1/u$, $t-u$ for $t+(-u)$, and freely use numerals n—abbreviating $\underbrace{1+\cdots+1}_{n\times}$—and exponentiation with integer exponents as in t^k. We shall further write

$$1_x \text{ for } \frac{x}{x} \qquad \text{and} \qquad 0_x \text{ for } 1-1_x,$$

so, $0_0 = 1_1 = 1$, $0_1 = 1_0 = 0$, and for all terms t,

$$0_t + 1_t = 1.$$

We write $\Sigma_r = (0,1,+,\cdot,-,)$ for the signature of rings. A *polynomial* is an expression over Σ_r, thus without inverse operator. Note that every polynomial can be represented as a sum of *monomials*, i.e. products of variables with integer coefficients. Meadow terms enjoy a particular standard representation which was introduced in [7].

Definition 2.1. A term $t \in Ter(\Sigma_m, X)$ is a *Standard Meadow Form (SMF)* if, for some $n \in \mathbb{N}$, t is an *SMF of level* n. SMFs of level n are defined as follows:

1. *SMF of level* 0 : each expression of the form s/t with s and t ranging over polynomials,
2. *SMF of level* $n+1$: each expression of the form

   $$0_{t'} \cdot s + 1_{t'} \cdot t$$

   with t' ranging over polynomials and s and t over SMFs of level n.

Theorem 2.2. *For each $t \in Ter(\Sigma_m, X)$ there exist an SMF t_{SMF} with the same variables such that $Md \vdash t = t_{SMF}$.*

Proof. See [7]. □

It follows that every meadow term is provably equal to a sum of quotients of polynomials.

Corollary 2.3. *For every $t \in Ter(\Sigma_m, X)$ there exist polynomials $s_0, t_0, \ldots, s_n, t_n$ such that*

$$Md \vdash t = \frac{s_0}{t_0} + \ldots + \frac{s_n}{t_n}$$

Proof. Let t_{SMF} be a SMF of t. We employ induction on its level n. If $n = 0$ then $t_{SMF} = s_0/t_0$ with s_0 and t_0 polynomials. Assume $n = m+1$. Then $t_{SMF} = 0_{t''} \cdot s + 1_{t''} \cdot t'$ where t'' is a polynomial, and s, t' are SMF's of level m. By the induction hypothesis $s = s_0/t_0 + \ldots + s_k/t_k$ and $t' = u_0/v_0 + \ldots + u_l/v_l$ with $s_0, t_0, \ldots, s_k, t_k, u_1, v_1, \ldots, u_l, v_l$ polynomials. Then

$$\begin{array}{rcl}
t_{SMF} & = & 0_{t''} \cdot s + 1_{t''} \cdot t' \\
 & = & (1 - \frac{t''}{t''}) \cdot s + \frac{t''}{t''} \cdot t' \\
 & = & s - (\frac{t'' s_0}{t'' t_0} + \ldots + \frac{t'' s_k}{t'' t_k}) + \frac{t'' u_0}{t'' v_0} + \ldots + \frac{t'' u_l}{t'' v_l} \\
 & = & s + \frac{-t'' s_0}{t'' t_0} + \ldots + \frac{-t'' s_k}{t'' t_k} + \frac{t'' u_0}{t'' v_0} + \ldots + \frac{t'' u_l}{t'' v_l}
\end{array}$$

and the last term is again a sum of quotients of polynomials. □

The term *cancellation meadow* was introduced in [8] for a *zero-totalized field*—a field in which $0^{-1} = 0$. Cancellation meadows satisfy in addition the so-called *cancellation axiom*

$$x \neq 0 \ \&\ x \cdot y = x \cdot z \ \longrightarrow\ y = z.$$

An equivalent version of the cancellation axiom is the *Inverse Law* (*IL*), i.e., the conditional axiom

$$x \neq 0 \ \longrightarrow\ x \cdot x^{-1} = 1. \quad (IL)$$

So *IL* states that there are no proper zero divisors. (Another equivalent formulation of the cancellation property is $x \cdot y = 0 \ \longrightarrow\ x = 0$ or $y = 0$.) The rationals $\mathbb{Q}$ form a cancellation meadow, $\mathbb{Z}/6\mathbb{Z}$ does not.

3 Calculation on cancellation meadows

Instruction sequences for functions on the rational numbers are designed in such a way that computations can be performed only with the aid of auxiliary variables to which initially the input values are copied, and from which the final values are copied to the output.

Definition 3.1. 1. We distinguish two infinite, countable sets of input and auxiliary variables $\mathit{Var}_{in} = \{x_i \mid i \in \mathbb{N}\}$ and $\mathit{Var}_{aux} = \{a_i \mid i \in \mathbb{N}\}$, and a single output variable y. Var denotes the union of these variables.

2. The instruction set $\mathit{Ins}(\mathbb{Q})$—instructions on the rational numbers—consists of the following input, auxiliary and output instructions:

$$\begin{array}{rcl}
\mathit{Ins}(\mathbb{Q})_{in} & = & \{a.\mathtt{cp}(x) \mid a \in \mathit{Var}_{aux} \ \&\ x \in \mathit{Var}_{in}\}, \\[1em]
\mathit{Ins}(\mathbb{Q})_{aux} & = & \{a.\mathtt{set{:}0}, a.\mathtt{set{:}1}, a.\mathtt{set{:}ai}, a.\mathtt{set{:}mi} \mid a \in \mathit{Var}_{aux}\}, \\
& & \cup\{a.\mathtt{set{:}a}(a'), a.\mathtt{set{:}m}(a'), \mid a, a' \in \mathit{Var}_{aux}\} \\
& & \cup\{a.\mathtt{test{:}0} \mid a \in \mathit{Var}_{aux}\} \\[1em]
\mathit{Ins}(\mathbb{Q})_{out} & = & \{y.\mathtt{cp}(a) \mid a \in \mathit{Var}_{aux}\}.
\end{array}$$

Here `ai` and `mi` refer to the unary meadow operations of additive and multiplicative inversion, and `a` and `m` to binary addition and multiplication. The intended meaning of these instructions is depicted in Table 4. Since assignment instructions always succeed, it is assumed that the returned truth value is `true`. An instruction of the form $a.\mathtt{test{:}0}$ is not an assignment instruction but a zero test and returns a truth value depending on the value of a.

$a.\mathtt{cp}(x)$	$[\ a \Leftarrow x\ ]$
$a.\mathtt{set{:}0}$	$[\ a \Leftarrow 0\ ]$
$a.\mathtt{set{:}1}$	$[\ a \Leftarrow 1\ ]$
$a.\mathtt{set{:}ai}$	$[\ a \Leftarrow -a\ ]$
$a.\mathtt{set{:}mi}$	$[\ a \Leftarrow a^{-1}\ ]$
$a.\mathtt{set{:}a}(a')$	$[\ a \Leftarrow a + a'\ ]$
$a.\mathtt{set{:}m}(a')$	$[\ a \Leftarrow a \cdot a'\ ]$
$a.\mathtt{test{:}0}$	
$y.\mathtt{cp}(a)$	$[\ y \Leftarrow a\ ]$

Table 4: The instruction set and its informal semantics

Examples 3.2. 1. Consider the following straight-line instruction sequence I_1:

$$a_0.\mathtt{cp}(x_0); a_1.\mathtt{set{:}1}; a_1.\mathtt{set{:}a}(a_1); a_0.\mathtt{set{:}a}(a_1);$$
$$a_1.\mathtt{cp}(x_0); a_0.\mathtt{set{:}m}(a_1); a_0.\mathtt{set{:}mi}; y.\mathtt{cp}(a_0); !$$

I_1 represents the total meadow mapping $x \mapsto ((x+2)x)^{-1}$: first the auxiliary variable a_0 is assigned the value of the input variable x_0 and then is raised by 2, after which a_0 is multiplied by x_0, inverted and copied to the output variable y.

2. The periodic instruction sequence I_2

$$\begin{aligned}&a_0.\texttt{cp}(x_0); a_1.\texttt{cp}(x_1); a_2.\texttt{set:1}; a_3.\texttt{set:1}; a_3.\texttt{set:ai};\\ &(-a_1.\texttt{test:0}; \#3; y.\texttt{cp}(a_2); !; a_2.\texttt{set:m}(a_0); a_1.\texttt{set:a}(a_3))^\omega\end{aligned}$$

   represents the partial mapping $(x_0, x_1) \mapsto x_0^{x_1}$: first, the two arguments are copied to the auxiliary variables a_0 and a_1, and a_2 and a_3 are assigned the constants 1 and -1, respectively. In the repetition, a_2 is multiplied by the first argument, and the second argument is decreased by 1 until the zero test succeeds and the value of a_2 is copied to the output. This partial meadow mapping is defined for all pairs of the form $\langle x, n\rangle$.

Meadows are standard mathematical structures, and as such, they may be described using standard logical formalisms. Here, we shall use the following first-order predicate logic over meadows and regular threads consisting of

1. the constants 0 and 1,
2. countably infinite constants $c_0, c_1, \ldots$,
3. the unary function symbols $-$ and ${}^{-1}$, representing additive and multiplicative inversion,
4. the binary function symbols $+$ and $\cdot$, written infix and representing addition and multiplication,
5. for every regular thread T and $k, n \in \mathbb{N}$, a $k+1$-ary termination predicate $R_{T,k,n}(\vec{x})$, describing the property T *terminates on input* $x_0, x_1, \ldots, x_k$ *after at most* n *steps*,
6. the usual Boolean connectives and first-order quantifiers with variables ranging over elements of meadows.

The standard interpretation of the termination predicates is given below. Since assignment instructions always succeed, we may assume that regular threads corresponding to instruction sequences on rational numbers are of the form S, D, $T \trianglelefteq a_i.\texttt{test:0} \trianglerighteq T'$ or $\mathit{ins} \circ T$ where ins is an assignment instruction.

Definition 3.3. Let $\mathcal{M}$ be a meadow.

1. If α is an assignment in $\mathcal{M}$, i.e. $\alpha \in \mathcal{M}^{\mathit{Var}}$, $v \in \mathit{Var}$, and $m \in \mathcal{M}$, we denote by $\alpha[v := m]$ the assignment α' with

$$\alpha'(v') = \begin{cases} m & \text{if } v' \equiv v \\ \alpha(v') & \text{otherwise.} \end{cases}$$

2. Let T be a regular thread and $n \in \mathbb{N}$. $R^{\mathcal{M}}_{T,n} \subseteq \mathcal{M}^{\mathit{Var}}$ is defined inductively as follows.

(a)

$$R^{\mathcal{M}}_{T,0} = \begin{cases} \mathcal{M}^{\mathit{Var}} & \text{if } T = S, \\ \emptyset & \text{otherwise.} \end{cases}$$

(b)

$$\begin{array}{rcl}
R^{\mathcal{M}}_{S,n+1} & = & \mathcal{M}^{\mathit{Var}} \\
R^{\mathcal{M}}_{D,n+1} & = & \emptyset \\
R^{\mathcal{M}}_{a_i.\texttt{cp}(x_j)\circ T,n+1} & = & \{\alpha \in \mathcal{M}^{\mathit{Var}} \mid \alpha[a_i := \alpha(x_j)] \in R^{\mathcal{M}}_{T,n}\} \\
R^{\mathcal{M}}_{a_i.\texttt{set:0}\circ T,n+1} & = & \{\alpha \in \mathcal{M}^{\mathit{Var}} \mid \alpha[a_i := 0] \in R^{\mathcal{M}}_{T,n}\} \\
R^{\mathcal{M}}_{a_i.\texttt{set:1}\circ T,n+1} & = & \{\alpha \in \mathcal{M}^{\mathit{Var}} \mid \alpha[a_i := 1] \in R^{\mathcal{M}}_{T,n}\} \\
R^{\mathcal{M}}_{a_i.\texttt{set:ai}\circ T,n+1} & = & \{\alpha \in \mathcal{M}^{\mathit{Var}} \mid \alpha[a_i := -\alpha(a_i)] \in R^{\mathcal{M}}_{T,n}\} \\
R^{\mathcal{M}}_{a_i.\texttt{set:mi}\circ T,n+1} & = & \{\alpha \in \mathcal{M}^{\mathit{Var}} \mid \alpha[a_i := \alpha(a_i)^{-1}] \in R^{\mathcal{M}}_{T,n}\} \\
R^{\mathcal{M}}_{a_i.\texttt{set:a}(a_j)\circ T,n+1} & = & \{\alpha \in \mathcal{M}^{\mathit{Var}} \mid \alpha[a_i := \alpha(a_i) + \alpha(a_j)] \in R^{\mathcal{M}}_{T,n}\} \\
R^{\mathcal{M}}_{a_i.\texttt{set:m}(a_j)\circ T,n+1} & = & \{\alpha \in \mathcal{M}^{\mathit{Var}} \mid \alpha[a_i := \alpha(a_i) \cdot \alpha(a_j)] \in R^{\mathcal{M}}_{T,n}\} \\
R^{\mathcal{M}}_{T \trianglelefteq a_i.\texttt{test:0} \trianglerighteq T',n+1} & = & \{\alpha \in R^{\mathcal{M}}_{T,n} \mid \alpha(a_i) = 0\} \cup \{\alpha \in R^{\mathcal{M}}_{T',n} \mid \alpha(a_i) \neq 0\} \\
R^{\mathcal{M}}_{y.\texttt{cp}(a_j)\circ T,n+1} & = & \{\alpha \in \mathcal{M}^{\mathit{Var}} \mid \alpha[y := \alpha(a_j)] \in R^{\mathcal{M}}_{T,n}\}
\end{array}$$

3. For $k, n \in \mathbb{N}$ and regular thread T we define $[\![R_{T,k,n}]\!]_{\mathcal{M}} \subseteq \mathcal{M}^{k+1}$ by

$$[\![R_{T,k,n}]\!]_{\mathcal{M}} = \{\langle \alpha(x_0), \ldots, \alpha(x_k) \rangle \mid \alpha \in R^{\mathcal{M}}_{T,n}\}.$$

Example 3.4. We consider once more the instruction sequences I_1 and I_2 introduced in Example 3.2.

1. It is easy to see that, if $I = \mathit{ins}_1; \ldots; \mathit{ins}_n; !$ is a straight-line instruction sequence consisting of n assignment instructions and ending in a single final termination instruction, then $\mathcal{M} \models \forall x_0, \ldots, x_k \; R_{|I|,k,n}(x_0, \ldots, x_k)$ for every meadow $\mathcal{M}$. Hence, in particular, $\mathcal{M} \models \forall x \; R_{|I_1|,0,8}(x)$.

2. $|I_2|$ starts with 5 initializing actions and repeats 3 consecutive actions until the zero test succeeds in which case termination occurs after a final copying action. We thus have for all meadows $\mathcal{M}$, $\mathcal{M} \models \forall x \; R_{|I_2|,1,3n+7}(x, n)$.

Lemma 3.5. *For all $k, n \in \mathbb{N}$, regular threads T and meadows $\mathcal{M}$,*

1. $R^{\mathcal{M}}_{T,n} \subseteq R^{\mathcal{M}}_{T,n+1}$
2. $R^{\mathcal{M}}_{T,n} = R^{\mathcal{M}}_{\pi_{n+1}(T),n}$
3. $\mathcal{M} \models \forall x_0, \ldots, x_k \; (R_{T,k,n}(x_0, \ldots, x_k) \longrightarrow R_{T,k,n+1}(x_0, \ldots, x_k))$
4. $\mathcal{M} \models \forall x_0, \ldots, x_k \; (R_{T,k,n}(x_0, \ldots, x_k) \longleftrightarrow R_{\pi_{n+1}(T),k,n}(x_0, \ldots, x_k))$

Proof. (1) and (2) are proven by straightforward induction; (3) and (4) follow from (1) and (2), respectively. □

The *apply operator* has been introduced in [6] as a means to transform a given state machine according to a thread. Given a meadow, we view its assignments as state machines which can be transformed. The corresponding apply operator is then defined as follows.

Definition 3.6. Let $\mathcal{M}$ be a meadow and T be a finite thread. We define the apply operator $T\bullet : \mathcal{M}^{Var} \cup \{\mathsf{D}\} \to \mathcal{M}^{Var} \cup \{\mathsf{D}\}$ as follows.

$$\begin{array}{rcl}
T \bullet \mathsf{D} & = & \mathsf{D} \\
\mathsf{S} \bullet \alpha & = & \alpha \\
\mathsf{D} \bullet \alpha & = & \mathsf{D} \\
(a_i.\mathtt{cp}(x_j) \circ T) \bullet \alpha & = & T \bullet \alpha[a_i := \alpha(x_j)] \\
(a_i.\mathtt{set:0} \circ T) \bullet \alpha & = & T \bullet \alpha[a_i := 0] \\
(a_i.\mathtt{set:1} \circ T) \bullet \alpha & = & T \bullet \alpha[a_i := 1] \\
(a_i.\mathtt{set:ai} \circ T) \bullet \alpha & = & T \bullet \alpha[a_i := -\alpha(a_i)] \\
(a_i.\mathtt{set:mi} \circ T) \bullet \alpha & = & T \bullet \alpha[a_i := \alpha(a_i)^{-1}] \\
(a_i.\mathtt{set:a}(a_j) \circ T) \bullet \alpha & = & T \bullet \alpha[a_i := \alpha(a_i) + \alpha(a_j)] \\
(a_i.\mathtt{set:m}(a_j) \circ T) \bullet \alpha & = & T \bullet \alpha[a_i := \alpha(a_i) \cdot \alpha(a_j)] \\
(T \trianglelefteq a_i.\mathtt{test:0} \trianglerighteq T') \bullet \alpha & = & \begin{cases} T \bullet \alpha & \text{if } \alpha(a_i) = 0 \\ T' \bullet \alpha & \text{otherwise} \end{cases} \\
(y.\mathtt{cp}(a_j) \circ T) \bullet \alpha & = & T \bullet \alpha[y := \alpha(a_j)]
\end{array}$$

For infinite threads T, the apply operator is defined on certain inputs if T terminates after finitely many steps.

Lemma 3.7. *Let $\mathcal{M}$ be a meadow and T be a regular thread. Then for all $n \in \mathbb{N}$ and $\alpha \in R^{\mathcal{M}}_{T,n}$,*

1. *$\pi_{n+1}(T) \bullet \alpha \neq \mathsf{D}$, and*
2. *$\forall k > n\ \pi_k(T) \bullet \alpha = \pi_{n+1}(T) \bullet \alpha$.*

Proof. By straightforward induction. □

We can therefore define partial mappings corresponding to regular threads as below.

Definition 3.8. Let $\mathcal{M}$ be a meadow.

1. $\alpha \in \mathcal{M}^{Var}$ is called *initial* if $\alpha(v) = 0$ for all $v \in Var - Var_{in}$.
2. Let T be a regular thread and $k \in \mathbb{N}$. Then $[\![T]\!]^k_{\mathcal{M}} : \mathcal{M}^{k+1} \xrightarrow{p} \mathcal{M}$ denotes the partial mapping defined as follows:

   $$[\![T]\!]^k_{\mathcal{M}}(m_0, \ldots, m_k) = \begin{cases} (\pi_{n+1}(T) \bullet \alpha_{m_0,\ldots,m_k})(y) & \text{if } \alpha_{m_0,\ldots,m_k} \in R^{\mathcal{M}}_{T,n}, \\ \text{undefined} & \text{if for all } n \in \mathbb{N},\ \alpha_{m_0,\ldots,m_k} \notin R^{\mathcal{M}}_{T,n}. \end{cases}$$

   where $\alpha_{m_0,\ldots,m_k} \in \mathcal{M}^{Var}$ is the *initial* assignment with $\alpha(x_i) = m_i$ for $0 \leq i \leq k$ and $\alpha(v) = 0$ for $v \in Var - \{x_0, \ldots, x_k\}$.

Notation 3.9. If I is an instruction sequence we shall write $[\![I]\!]^k_{\mathcal{M}}$ for the corresponding meadow mapping instead of $[\![|I|]\!]^k_{\mathcal{M}}$. Moreover, when dealing with partial mappings, we let the symbols $\uparrow$ and $\downarrow$ denote un- and definedness, respectively.

Example 3.10. We consider again the instruction sequences I_1 and I_2 given in Example 3.2.

1. Observe that

$$\begin{aligned} |I_1| \quad &= \quad a_0.\mathtt{cp}(x_0) \circ a_1.\mathtt{set\!:\!1} \circ a_1.\mathtt{set\!:\!a}(a_1) \circ a_0.\mathtt{set\!:\!a}(a_1) \\ &\quad\ \circ a_1.\mathtt{cp}(x_0) \circ a_0.\mathtt{set\!:\!m}(a_1) \circ a_0.\mathtt{set\!:\!mi} \circ y.\mathtt{cp}(a_0) \circ \mathsf{S}. \end{aligned}$$

Thus

$$\begin{aligned} |I_1| \bullet \alpha \quad &= \quad \alpha[a_0 := \alpha(x_0)][a_1 := 1][a_1 := 2][a_0 := \alpha(x_0) + 2] \\ &\quad\ [a_1 := \alpha(x_0)][a_0 := (\alpha(x_0) + 2) \cdot \alpha(x_0)] \\ &\quad\ [a_0 := ((\alpha(x_0) + 2) \cdot \alpha(x_0))^{-1}][y := ((\alpha(x_0) + 2) \cdot \alpha(x_0))^{-1}] \end{aligned}$$

for every meadow $\mathcal{M}$ and every assignment $\alpha \in \mathcal{M}^{Var}$. Hence $[\![I_1]\!]^0_{\mathcal{M}}(m) = ((m + 2)m)^{-1}$.

2. The periodic thread $|I_2|$ satisfies the equations

$$\begin{aligned} |I_2| \quad &= \quad a_0.\mathtt{cp}(x_0) \circ a_1.\mathtt{cp}(x_1) \circ a_2.\mathtt{set\!:\!1} \circ a_3.\mathtt{set\!:\!1} \circ a_3.\mathtt{set\!:\!ai} \circ T \\ T \quad &= \quad (y.\mathtt{cp}(a_2) \circ S) \trianglelefteq a_1.\mathtt{test\!:\!0} \trianglerighteq (a_2.\mathtt{set\!:\!m}(a_0) \circ a_1.\mathtt{set\!:\!a}(a_3) \circ T). \end{aligned}$$

So

$$\begin{aligned} |I_2| \bullet \alpha \quad &= \quad T \bullet \alpha[a_0 := \alpha(x_0)][a_1 := \alpha(x_1)][a_2 := 1][a_3 := 1][a_3 := -1] \\ T \bullet \alpha \quad &= \quad \begin{cases} \alpha[y := \alpha(a_2)] & \text{if } \alpha(a_1) = 0, \\ T \bullet \alpha[a_2 := \alpha(a_2) \cdot \alpha(x_0)][a_1 := \alpha(x_1) - 1] & \text{otherwise.} \end{cases} \end{aligned}$$

It follows that $|I_2| \bullet \alpha \neq \mathsf{D}$ if and only if $\alpha(x_1) = n$ for some $n \in \mathbb{N}$. Hence

$$[\![I_2]\!]^1_{\mathcal{M}}(m_0, m_1) = \begin{cases} m_0^{m_1} & \text{if } m_1 = n \text{ for some } n \in \mathbb{N} \\ \uparrow & \text{otherwise} \end{cases}$$

for every meadow $\mathcal{M}$. In case $\mathcal{M} = \mathbb{Q}$, I_2 yields a non-total mapping; on prime fields $\mathbb{Z}/p\mathbb{Z}$—considered zero-totalized— this mapping is total.

Every meadow mapping that is total on all meadows is clearly total on all cancellation meadows. The converse, however, does not hold: consider the instruction sequence

$$I = a_0.\mathtt{cp}(x_0); -a_0.\mathtt{test\!:\!0}; \#2; \#4; a_0.\mathtt{set\!:\!a}(a_0); +a_0.\mathtt{test\!:\!0}; \#0; y.\mathtt{cp}(a_1); !.$$

Given any meadow $\mathcal{M}$, we have

$$[\![I]\!]^0_{\mathcal{M}}(m) = \begin{cases} 0 & \text{if } m = 0, \\ 0 & \text{if } m \neq 0 \ \&\ 2m \neq 0, \\ \uparrow & \text{otherwise.} \end{cases}$$

In the absence of proper zero divisors, $m = 0$ if $2m = 0$. Thus $[\![I]\!]^0_{\mathcal{M}}$ is the constant zero mapping on every cancellation meadow $\mathcal{M}$. On the zero-totalized field $\mathbb{Z}/6\mathbb{Z}$, however, $3 \neq 0$ and $2 \times 3 = 0$, and thus $[\![I]\!]^0_{\mathbb{Z}/6\mathbb{Z}}(3) \uparrow$.

4 Characterization of total calculation on cancellation meadows

In this section we shall prove the main theorem.

Total mappings share the following *finite representation property.*

Proposition 4.1. *Let T be a regular thread and $k \in \mathbb{N}$. If $[\![T]\!]^k_{\mathcal{M}}$ is total on all meadows $\mathcal{M}$, then there exists a finite thread T' such that $[\![T]\!]^k_{\mathcal{M}} = [\![T']\!]^k_{\mathcal{M}}$ for all meadows $\mathcal{M}$.*

Proof. Consider the set Γ consisting of all meadow axioms together with the infinite set $\{\neg R_{T,k,n}(c_0, \ldots, c_k) \mid n \in \mathbb{N}\}$. If Γ is finitely satisfiable, it must be simultaneously satisfiable, by Compactness, say in some meadow $\mathcal{M}$. This means that $[\![T]\!]^k_{\mathcal{M}}$ is not total, contradicting the assumption. We may therefore assume that Γ is not finitely satisfiable. By a standard model-theoretic argument and the monotonicity of the termination predicate (Lemma 3.5 (3)), it follows that for some $n \in \mathbb{N}$ and all meadows $\mathcal{M}$, $\mathcal{M} \models \forall x_0, \ldots, x_k\ R_{T,k,n}(x_0, \ldots, x_k)$. Hence $\mathcal{M} \models \forall x_0, \ldots, x_k\ R_{\pi_{n+1}(T),k,n}(x_0, \ldots, x_k)$ for all $\mathcal{M}$ by Lemma 3.5 (4). Then $[\![T]\!]^k_{\mathcal{M}} = [\![\pi_{n+1}(T)]\!]^k_{\mathcal{M}}$ for all meadows $\mathcal{M}$. □

Proposition 4.2. *Let T be a regular thread and $k \in \mathbb{N}$. If $[\![T]\!]^k_{\mathcal{M}}$ is total on all cancellation meadows $\mathcal{M}$, then there exists a finite thread T' such that $[\![T]\!]^k_{\mathcal{M}} = [\![T']\!]^k_{\mathcal{M}}$ for all cancellation meadows $\mathcal{M}$.*

Proof. Repeat the previous proof with Γ supplemented with the cancellation axiom

$$\forall x, y, z\ (x \neq 0\ \&\ x \cdot y = x \cdot z \longrightarrow y = z).$$

□

Next we shall show that tests can be abandoned without the loss of expressive power.

For $t \in Ter(\Sigma_m, Var)$, $[\![t]\!]_{\mathcal{M},\alpha}$ denotes the interpretation of t in the meadow $\mathcal{M}$ under the assignment α, and if $\sigma \in Ter(\Sigma_m, Var)^{Var}$, then t^σ is the result of substituting all variables v occurring in t by $\sigma(v)$. Recall that substitutions and assignments interact in the following way.

Lemma 4.3. *Let $\mathcal{M}$ be a meadow, $\alpha \in \mathcal{M}^{Var}$ an assignment and $\sigma \in Ter(\Sigma_m, Var)^{Var}$ a substitution. Define $\alpha' \in \mathcal{M}^{Var}$ by $\alpha'(v) = [\![v^\sigma]\!]_{\mathcal{M},\alpha}$. Then for all $t \in Ter(\Sigma_m, Var)$,*

$$[\![t]\!]_{\mathcal{M},\alpha'} = [\![t^\sigma]\!]_{\mathcal{M},\alpha}.$$

Proposition 4.4. *Let T be a finite thread and $k \in \mathbb{N}$. Then there exists a term $t_T \in Ter(\Sigma_m, \{x_0, \ldots, x_k\})$ such that for all cancellation meadows $\mathcal{M}$ and all $m_0, \ldots, m_k \in \mathcal{M}$,*

$$[\![T]\!]^k_{\mathcal{M}}(m_0, \ldots, m_k) \downarrow \longrightarrow [\![T]\!]^k_{\mathcal{M}}(m_0, \ldots, m_k) = [\![t_T]\!]_{\mathcal{M},\alpha_{m_0,\ldots,m_k}}.$$

Proof. We use induction loading and employ structural induction on T in order to prove the assertion stating the existence of a term $t_T \in \mathit{Ter}(\Sigma_m, \mathit{Var})$ such that for all cancellation meadows $\mathcal{M}$ and all assignments $\alpha \in \mathcal{M}^{\mathit{Var}}$,

$$(T \bullet \alpha)(y) = [\![t_T]\!]_{\mathcal{M},\alpha}$$

if $T \bullet \alpha \neq \mathsf{D}$.

If $T = \mathsf{S}$, then

$$\mathsf{S} \bullet \alpha)(y) = \alpha(y) = [\![y]\!]_{\mathcal{M},\alpha}.$$

Hence $t_{\mathsf{S}} \equiv y$. If $T = \mathsf{D}$, we also put $t_{\mathsf{D}} \equiv y$. For the induction step, we have to distinguish 9 cases each of which corresponds to one the 9 instructions sorts in $\mathit{Ins}(\mathbb{Q})$. The assignment instructions are proven straightforwardly using the previous substitution lemma. We show 3 cases.

Suppose $T = a_i.\mathtt{cp}(x_j) \circ T'$ and $T \bullet \alpha \neq \mathsf{D}$. Then

$$\begin{array}{rcll} (T \bullet \alpha)(y) & = & (T' \bullet \alpha[a_i := \alpha(x_j)])(y) & \\ & = & [\![t_{T'}]\!]_{\mathcal{M},\alpha[a_i:=\alpha(x_j)]} & \text{by the induction hypothesis} \\ & = & [\![t^{\sigma}_{T'}]\!]_{\mathcal{M},\alpha} & \text{by Lemma 4.3} \end{array}$$

where $\sigma(a_i) = x_j$, and $\sigma(v) = v$ if $v \not\equiv a_i$. Hence $t_T \equiv t^{\sigma}_{T'}$ suffices. Likewise, if $T = a_i.\mathtt{set:a}(a_j) \circ T'$ and $T \bullet \alpha \neq \mathsf{D}$, then

$$\begin{array}{rcll} (T \bullet \alpha)(y) & = & (T' \bullet \alpha[a_i := \alpha(a_i) + \alpha(a_j)])(y) & \\ & = & [\![t_{T'}]\!]_{\mathcal{M},\alpha[a_i:=\alpha(a_i)+\alpha(a_j)]} & \text{by the induction hypothesis} \\ & = & [\![t^{\sigma}_{T'}]\!]_{\mathcal{M},\alpha} & \text{by Lemma 4.3} \end{array}$$

where $\sigma(a_i) = a_i + a_j$, and $\sigma(v) = v$ if $v \not\equiv a_i$. And if $T = y.\mathtt{cp}(a_j) \circ T'$ and $T \bullet \alpha \neq \mathsf{D}$, then

$$\begin{array}{rcll} (T \bullet \alpha)(y) & = & (T' \bullet \alpha[y := \alpha(a_j)])(y) & \\ & = & [\![t_{T'}]\!]_{\mathcal{M},\alpha[y:=\alpha(a_j)]} & \text{by the induction hypothesis} \\ & = & [\![t^{\sigma}_{T'}]\!]_{\mathcal{M},\alpha} & \text{by Lemma 4.3} \end{array}$$

where $\sigma(y) = a_j$, and $\sigma(v) = v$ if $v \not\equiv y$.

The case for the zero test exploits the fact that in every cancellation meadow $\mathcal{M}$ we have

$$[\![0_{a_i} \cdot t + 1_{a_i} \cdot t']\!]_{\mathcal{M},\alpha} = \begin{cases} [\![t]\!]_{\mathcal{M},\alpha} & \text{if } \alpha(a_i) = 0 \\ [\![t']\!]_{\mathcal{M},\alpha} & \text{otherwise} \end{cases}$$

Hence, if $T = T' \trianglelefteq a_i.\mathtt{test:0} \trianglerighteq T''$ we can take

$$t_T \equiv 0_{a_i} \cdot t_{T'} + 1_{a_i} \cdot t_{T''}.$$

The original assertion now follows from the observation that we can replace all occurrences of auxiliary variables, the output variable y and all input variables x_n with $k < n$ in the term t_T by 0 if α is initial. □

Definition 4.5. We shall say that the thread T *computes* $t \in \mathit{Ter}(\Sigma_m, \{x_0, \dots, x_k\})$, if for all cancellation meadows $\mathcal{M}$ and all $m_0, \dots, m_k \in \mathcal{M}$,

$$[\![T]\!]^k_{\mathcal{M}}(m_0, \dots, m_k) = [\![t]\!]_{\mathcal{M}, \alpha_{m_0, \dots, m_k}}.$$

Thus if T is finite, the free variables of t_T are among $\{x_0, \dots, x_k\}$ and $[\![T]\!]^k_{\mathcal{M}}$ is total, then T computes the term t_T. Conversely, every meadow term t with free variables in Var_{in} can be computed by a finite thread T_t which is in addition *test-free*—that is, postconditional composition occurs as action prefix only—and which uses at most 5 auxiliary variables. To these ends, we shall define the *raise* T^1 *of a thread* T as the thread T' obtained from T by raising the subscript of every auxiliary variable occurring in T by 1.

Definition 4.6. 1. Let $i \in \mathit{Ins}(\mathbb{Q})$ be an instruction. Then i^1 is defined by

$$\begin{aligned}
a_i.\mathtt{cp}(x_j)^1 &= a_{i+1}.\mathtt{cp}(x_j)\\
a_i.\mathtt{set{:}0}^1 &= a_{i+1}.\mathtt{set{:}0}\\
a_i.\mathtt{set{:}1}^1 &= a_{i+1}.\mathtt{set{:}1}\\
a_i.\mathtt{set{:}ai}^1 &= a_{i+1}.\mathtt{set{:}ai}\\
a_i.\mathtt{set{:}mi}^1 &= a_{i+1}.\mathtt{set{:}mi}\\
a_i.\mathtt{set{:}a}(a_j)^1 &= a_{i+1}.\mathtt{set{:}a}(a_{j+1})\\
a_i.\mathtt{set{:}m}(a_j)^1 &= a_{i+1}.\mathtt{set{:}m}(a_{j+1})\\
a_i.\mathtt{test{:}0}^1 &= a_{i+1}.\mathtt{test{:}0}\\
y.\mathtt{cp}(a_j)^1 &= y.\mathtt{cp}(a_{j+1})
\end{aligned}$$

2. Let T be a finite thread. Then T^1 is defined inductively by $\mathsf{S}^1 = \mathsf{S}$, $\mathsf{D}^1 = \mathsf{D}$, and $(T' \trianglelefteq i \trianglerighteq T'')^1 = T'^1 \trianglelefteq i^1 \trianglerighteq T''^1$ for $i \in \mathit{Ins}(\mathbb{Q})$.

A thread and its raise compute the same values.

Lemma 4.7. *Let T be a finite thread and $k \in \mathbb{N}$. Then for all meadows $\mathcal{M}$ and all $m_0, \dots, m_k \in \mathcal{M}$*

$$[\![T]\!]^k_{\mathcal{M}}(m_0, \dots, m_k)\downarrow \longrightarrow [\![T]\!]^k_{\mathcal{M}}(m_0, \dots, m_k) = [\![T^1]\!]^k_{\mathcal{M}}(m_0, \dots, m_k).$$

Proof. For the sake of the proof, we define for $\alpha \in \mathcal{M}^{\mathit{Var}}$ the raise $\alpha^1 \in \mathcal{M}^{\mathit{Var}}$ by

$$\alpha^1(v) = \begin{cases} \alpha(a_{i+1}) & \text{if } v \equiv a_i,\\ \alpha(v) & \text{if } v \notin \mathit{Var}_{aux}.\end{cases}$$

We now show that

$$T \bullet \alpha^1 \neq \mathsf{D} \longrightarrow (T \bullet \alpha^1)(y) = (T^1 \bullet \alpha)(y)$$

by structural induction on T. If $T = \mathsf{S}$, then

$$(T \bullet \alpha^1)(y) = (\mathsf{S} \bullet \alpha^1)(y) = \alpha^1(y) = \alpha(y) = (\mathsf{S} \bullet \alpha)(y) = (T^1 \bullet \alpha)(y).$$

For the induction step, we have to distinguish 9 cases each of which corresponds to one of the 9 instruction sorts. Each case follows straightforwardly. We show the case that $T = a_i.\texttt{cp}(x_j) \circ T'$:

$$\begin{array}{rcl}
(T \bullet \alpha^1)(y) & = & ((a_i.\texttt{cp}(x_j) \circ T') \bullet \alpha^1)(y) \\
& = & (T' \bullet \alpha^1[a_i := \alpha^1(x_j)])(y) \\
& = & (T' \bullet (\alpha[a_{i+1} := \alpha(x_j)])^1)(y) \\
& = & (T'^1 \bullet \alpha[a_{i+1} := \alpha(x_j)])(y) \text{ by the induction hypothesis} \\
& = & (a_{i+1}.\texttt{cp}(x_j) \circ T'^1 \bullet \alpha)(y) \\
& = & (T^1 \bullet \alpha)(y)
\end{array}$$

The statement now follows from the the observation that $\alpha^1 = \alpha$ if α is initial. □

In the sequel, we shall say that a thread T (an instruction sequence I) *uses the auxiliary variable* a_i, if the variable a_i occurs in at least one of T's (I's) atomic actions (basic instructions). Moreover, we shall say that T (I) *uses* n *auxiliary variables*, if T (I) uses precisely the auxiliary variables $a_0, \ldots, a_{n-1}$.

Lemma 4.8. *1. If $t \in Var_{in} \cup \{0, 1\}$, then t can be computed by a finite and test-free thread that uses 1 auxiliary variable.*

2. If t can be computed by a finite and test-free thread that uses n auxiliary variables, then so can $-t$ and t^{-1}.

3. Suppose $t, t' \in Ter(\Sigma_m, Var_{in})$ can be computed by finite and test-free threads that use n and m auxiliary variables, respectively. If $n = m$, then $t+t'$ and $t\cdot t'$ can be computed by finite and test-free threads that use $n + 1$ auxiliary variables, and if $n \neq m$, then $t+t'$ and $t\cdot t'$ can be computed finite and test-free threads that use $\max\{n, m\}$ *auxiliary variables.*

Proof. We shall construct appropriate threads of the form

$$i_1 \circ \cdots \circ i_k \circ y.\texttt{cp}(a_0) \circ \mathsf{S}$$

1. Observe that $a_0.\texttt{cp}(x_i) \circ y.\texttt{cp}(a_0) \circ \mathsf{S}$, $a_0.\texttt{set:0} \circ y.\texttt{cp}(a_0) \circ \mathsf{S}$, and $a_0.\texttt{set:1} \circ y.\texttt{cp}(a_0) \circ \mathsf{S}$ compute x_i, 0 and 1, respectively.

2. Suppose $T = i_1 \circ \cdots \circ i_k \circ y.\texttt{cp}(a_0) \circ \mathsf{S}$ computes t. Then

   $$i_1 \circ \cdots \circ i_k \circ a_0.\texttt{set:ai} \circ y.\texttt{cp}(a_0) \circ \mathsf{S}$$

   computes $-t$ and

   $$i_1 \circ \cdots \circ i_k \circ a_0.\texttt{set:mi} \circ y.\texttt{cp}(a_0) \circ \mathsf{S}$$

   computes t^{-1}. Both threads use as many auxiliary variables as T and are finite and test-free.

3. Suppose $T = i_1 \circ \cdots \circ i_k \circ y.\mathtt{cp}(a_0) \circ \mathsf{S}$ uses n auxiliary variables to compute t, and $T' = j_1 \circ \cdots \circ j_l \circ y.\mathtt{cp}(a_0) \circ \mathsf{S}$ uses m auxiliary variables to compute t'. We first assume that $n \leq m$. Since by the previous lemma T^1 also computes t, we have that

   $$j_1 \circ \cdots \circ j_l \circ a_1.\mathtt{set:0} \circ \cdots \circ a_n.\mathtt{set:0} \circ i_1^1 \circ \cdots \circ i_k^1 \circ a_0.\mathtt{set:a}(a_1) \circ y.\mathtt{cp}(a_0) \circ \mathsf{S}$$

   computes $t + t'$. This thread is finite and test-free, and uses $n + 1$ auxiliary variables if $n = m$ and otherwise m variables. If $m < n$ then

   $$i_1 \circ \cdots \circ j_k \circ a_1.\mathtt{set:0} \circ \cdots \circ a_m.\mathtt{set:0} \circ j_1^1 \circ \cdots \circ j_l^1 \circ a_0.\mathtt{set:a}(a_1) \circ y.\mathtt{cp}(a_0) \circ \mathsf{S}$$

   uses n auxiliary variables and computes $t' + t$ and hence $t + t'$.

   For $t \cdot t'$ we replace the action $a_0.\mathtt{set:a}(a_1)$ in the above threads by $a_0.\mathtt{set:m}(a_1)$.

□

Proposition 4.9. *Let $t \in Ter(\Sigma_m, Var_{in})$ be a meadow term. Then t can be computed by a finite, test-free thread T that uses 5 auxiliary variables.*

Proof. By Corollary 2.3 t can be represented as a sum of quotients of polynomials. Moreover, every polynomial can be written as a sum of monomials, i.e. expressions of the form $n \cdot x_{i_1} \cdots x_{i_k}$ or $-n \cdot x_{i_1} \cdots x_{i_k}$. Since $n = 1 + \cdots + 1 + 0 + 0$ it can be computed by a finite and test-free thread that uses 2 auxiliary variables by 4.8.(1) and (3). Thus also $n \cdot x_{i_1}, \ldots, n \cdot x_{i_1} \cdots x_{i_k}$ can all be computed by finite and test-free threads that use 2 auxiliary variables. And the same holds for $-n \cdot x_{i_1} \cdots x_{i_k}$ by 4.8.(2). Thus every monomial can be computed by a finite and test-free thread that uses 2 auxiliary variables. It follows that every sum of monomials—and hence every polynomial—can be computed by a finite and test-free thread that uses 3 auxiliary variables by 4.8.(3). Whence every quotient of polynomials can be computed by a finite and test-free thread that uses 4 auxiliary variables by 4.8.(2) and (3). Invoking again 4.8.(3) we obtain that every sum of quotients of polynomials—and therefore t—can be computed by a finite and test-free thread that uses 5 auxiliary variables. □

Summarizing we have proven the following completeness result.

Theorem 4.10. *Let I be an instruction sequence and $k \in \mathbb{N}$ be such that $[\![I]\!]^k_{\mathcal{M}}$ is a total mapping on all cancellation meadows $\mathcal{M}$. Then there exists a straight-line instruction sequence J which uses at most 5 auxiliary variables such that $[\![I]\!]^k_{\mathcal{M}} = [\![J]\!]^k_{\mathcal{M}}$ for all cancellation meadows $\mathcal{M}$.*

Proof. Suppose that $[\![I]\!]^k_{\mathcal{M}}$ is total on all cancellation meadows $\mathcal{M}$. By Proposition 4.2, we can pick a finite thread T such that $[\![I]\!]^k_{\mathcal{M}} = [\![T]\!]^k_{\mathcal{M}}$ for all cancellation meadows $\mathcal{M}$. By Proposition 4.4 we may assume that T computes the term $t \in Ter(\Sigma_m, \{x_0, \ldots, x_k\})$ which in turn is computed by a finite and test-free thread T' that uses 5 auxiliary variables by the previous proposition. We can now take a straight-line instruction sequence J with $|J| = T'$. □

5 Calculation on signed cancellation meadows

We obtain *signed meadows* by extending the signature Σ_m of meadows with the unary sign function $\mathbf{s}(\_)$. We write Σ_{ms} for this extended signature, so $\Sigma_{ms} = (0, 1, +, \cdot, -, {}^{-1}, \mathbf{s})$. The sign function $\mathbf{s}$ presupposes an ordering $<$ of its domain and is defined as follows:

$$\mathbf{s}(x) = \begin{cases} -1 & \text{if } x < 0, \\ 0 & \text{if } x = 0, \\ 1 & \text{if } x > 0. \end{cases}$$

One can define $\mathbf{s}$ in an equational manner by the set *Signs* of axioms given in Table 5. First, notice that by *Md* and axiom (1) (or axiom (2)) we find

$$\mathbf{s}(1_x) = 1_x \tag{1}$$
$$\mathbf{s}(0_x) = 0_x \tag{2}$$
$$\mathbf{s}(-1) = -1 \tag{3}$$
$$\mathbf{s}(x^{-1}) = \mathbf{s}(x) \tag{4}$$
$$\mathbf{s}(x \cdot y) = \mathbf{s}(x) \cdot \mathbf{s}(y) \tag{5}$$
$$0_{\mathbf{s}(x)-\mathbf{s}(y)} \cdot (\mathbf{s}(x+y) - \mathbf{s}(x)) = 0 \tag{6}$$

Table 5: The set *Signs* of axioms for the sign function

$$\mathbf{s}(0) = 0 \quad \text{and} \quad \mathbf{s}(1) = 1.$$

Then, observe that in combination with the inverse law *IL*, axiom (6) is an equational representation of the conditional equational axiom

$$\mathbf{s}(x) = \mathbf{s}(y) \quad \longrightarrow \quad \mathbf{s}(x+y) = \mathbf{s}(x).$$

The initial algebra of $Md \cup Signs$ is $\mathbb{Q}$ expanded with the sign function. A proof follows immediately from the techniques used in [9, 10].

Some consequences of the $Md \cup Signs$ are:

$$\mathbf{s}(x^2) = 1_x \text{ because } \mathbf{s}(x^2) = \mathbf{s}(x) \cdot \mathbf{s}(x) = \mathbf{s}(x) \cdot \mathbf{s}(x^{-1}) = \mathbf{s}(1_x) = 1_x, \tag{7}$$
$$\mathbf{s}(x^3) = \mathbf{s}(x) \text{ because } \mathbf{s}(x^3) = \mathbf{s}(x) \cdot \mathbf{s}(x) \cdot \mathbf{s}(x^{-1}) = \mathbf{s}(x \cdot (x \cdot x^{-1})) = \mathbf{s}(x), \tag{8}$$
$$1_x \cdot \mathbf{s}(x) = \mathbf{s}(x) \text{ because } 1_x \cdot \mathbf{s}(x) = \mathbf{s}(x^2) \cdot \mathbf{s}(x) = \mathbf{s}(x^3) = \mathbf{s}(x), \tag{9}$$
$$\begin{aligned} \mathbf{s}(x)^{-1} = \mathbf{s}(x) \text{ because } \mathbf{s}(x)^{-1} &= (\mathbf{s}(x)^2 \cdot \mathbf{s}(x)^{-1})^{-1} = (\mathbf{s}(x^2) \cdot \mathbf{s}(x)^{-1})^{-1} \\ &= (1_x \cdot \mathbf{s}(x)^{-1})^{-1} = 1_x \cdot \mathbf{s}(x) = \mathbf{s}(x). \end{aligned} \tag{10}$$

So, $0 = \mathbf{s}(x) - \mathbf{s}(x) = \mathbf{s}(x) - \mathbf{s}(x)^3 = \mathbf{s}(x)(1 - \mathbf{s}(x)^2)$ and hence

$$\mathbf{s}(x) \cdot (1 - \mathbf{s}(x)) \cdot (1 + \mathbf{s}(x)) = 0. \tag{11}$$

The *finite basis result* for the equational theory of cancellation meadows is formulated in a generic way so that it can be used for any expansion of a meadow that satisfies the propagation properties defined below.

Definition 5.1. Let Σ be an extension of $\Sigma_m = (0, 1, +, \cdot, -, ^{-1})$, the signature of meadows. Let $E \supseteq Md$ (with Md the set of axioms for meadows given in Table 3).

1. (Σ, E) has the **propagation property for pseudo units** if for each pair of Σ-terms t, r and context $C[\ ]$,
$$E \vdash 1_t \cdot C[r] = 1_t \cdot C[1_t \cdot r].$$

2. (Σ, E) has the **propagation property for pseudo zeros** if for each pair of Σ-terms t, r and context $C[\ ]$,
$$E \vdash 0_t \cdot C[r] = 0_t \cdot C[0_t \cdot r].$$

Preservation of these propagation properties admits the following nice result:

Theorem 5.2 (Generic Basis Theorem for Cancellation Meadows)**.** *If $\Sigma \supseteq \Sigma_m,\ E \supseteq Md$ and (Σ, E) has the pseudo unit and the pseudo zero propagation property, then E is a basis (a complete axiomatisation) of $Mod_\Sigma(E \cup IL)$.*

Bergstra and Ponse [7] proved that Md and $Md \cup Signs$ satisfy both propagation properties and are therefore complete axiomatizations of $Mod_\Sigma(Md \cup IL)$ and $Mod_\Sigma(Md \cup Signs \cup IL)$, respectively. Since

$$Md \cup Signs \cup IL \vdash t = 0_{t'} \cdot s + 1_{t'} \cdot s' \longrightarrow \mathbf{s}(t) = 0_{t'} \cdot \mathbf{s}(s) + 1_{t'} \cdot \mathbf{s}(s')$$

using IL and the axioms (1), (2) and (5) of *Signs*, it then follows that

$$Md \cup Signs \vdash t = 0_{t'} \cdot s + 1_{t'} \cdot s' \Longrightarrow Md \cup Signs \vdash \mathbf{s}(t) = 0_{t'} \cdot \mathbf{s}(s) + 1_{t'} \cdot \mathbf{s}(s'). \quad (\dagger)$$

We can hence adapt the Standard Meadow Form to signed meadow terms as follows.

We write $\Sigma_{rs} = (0, 1, +, \cdot, -, \mathbf{s})$ for the signature of signed rings. A *signed polynomial* is then an expression over Σ_{rs}, thus without inverse operator.

Definition 5.3. A term $t \in Ter(\Sigma_{ms}, X)$ is a *Standard Signed Meadow Form (SSMF)* if, for some $n \in \mathbb{N}$, t is an *SSMF of level n*. SSMFs of level n are defined as follows:

1. *SSMF of level* 0 : each expression of the form s/t with s and t ranging over signed polynomials,

2. *SSMF of level* $n + 1$: each expression of the form
$$0_{t'} \cdot s + 1_{t'} \cdot t$$
with t' ranging over signed polynomials and s and t over SSMFs of level n.

Theorem 5.4. *For each $t \in Ter(\Sigma_{ms}, X)$ there exist an SSMF t_{SSMF} with the same variables such that $Md \cup Signs \vdash t = t_{SSMF}$.*

Proof. As in [7] using (†). □

As in Corollary 2.3 it follows that every signed meadow term is provably equal to a sum of quotients of signed polynomials.

Corollary 5.5. *For every $t \in Ter(\Sigma_{ms}, X)$ there exist signed polynomials $s_0, t_0, \ldots, s_n, t_n$ such that*

$$Md \cup Signs \vdash t = \frac{s_0}{t_0} + \ldots + \frac{s_n}{t_n}.$$

Signed polynomials also enjoy a standard form.

Lemma 5.6. *Let t be a signed polynomial and $n \in \mathbb{N}$ be the number of its subterms of the form $\mathbf{s}(t')$. Then there are polynomials $t_1, t_{1_1}, \ldots, t_{1_n}, \ldots, t_i, t_{i_1}, \ldots, t_{i_n}, \ldots, t_{3n}, t_{3n_1}, \ldots, t_{3n_n}$ such that*

$$Md \cup Signs \vdash t = \Sigma_{i=1}^{3^n} \Pi_{j=1}^{n} 0_{\phi(\mathbf{s}(t_{i_j}))} \cdot t_i$$

where $\phi(\mathbf{s}(t_{i_j})) \in \{\mathbf{s}(t_{i_j}), 1 + \mathbf{s}(t_{i_j}), 1 - \mathbf{s}(t_{i_j})\}$.

Proof. We employ induction on the number n of subterms of the form $\mathbf{s}(t')$. If $n = 0$ then t itself is a polynomial and hence $t_1 \equiv t$ suffices.

Suppose $n = l + 1$ and pick an innermost subterm $\mathbf{s}(t')$ of t. Then $t \equiv C[\mathbf{s}(t')]$ for some context C and polynomial t'. From *IL* together with (11) it follows that $\mathbf{s}(t') = 0$ or $\mathbf{s}(t') = 1$ or $\mathbf{s}(t') = -1$. Thus

$$Md \cup Signs \vdash t = 0_{\mathbf{s}(t')} \cdot C[0] + 0_{1-\mathbf{s}(t')} \cdot C[1] + 0_{1+\mathbf{s}(t')} \cdot C[-1]$$

with $C[0], C[1]$, and $C[-1]$ having l signed subterms. We can now apply the induction hypothesis. □

A suitable instruction for computations on signed meadows is a.`set:s` with Boolean reply `true` and the obvious semantics $a \Leftarrow \mathbf{s}(a)$. We add this instruction to $Ins(\mathbb{Q})$, and consider instruction sequences and corresponding threads over the enriched instruction set in the sequel.

Example 5.7. Notice that, with the sign function available, the function $\max(x_0, x_1)$ has the following simple definition

$$\max(x_0, x_1) = \begin{cases} (\mathbf{s}(x_0) + 1) \cdot x_0/2 & \text{if } x_1 = 0 \\ \max(x_0 - x_1, 0) + x_1 & \text{otherwise.} \end{cases}$$

$\max(x, y)$ can be computed by the periodic instruction sequence

$$\begin{aligned}
&a_0.\texttt{cp}(x_0); a_1.\texttt{cp}(x_1); a_2.\texttt{set:0}; a_4.\texttt{cp}(x_0); a_1.\texttt{test:0}; \#2; \#11;\\
&(a_3.\texttt{set:1}; a_4.\texttt{set:s}; a_4.\texttt{set:a}(a_3); a_0.\texttt{set:m}(a_4);\\
&\ a_3.\texttt{set:a}(a_3); a_3.\texttt{set:mi}; a_0.\texttt{set:m}(a_3); a_0.\texttt{set:a}(a_2); y_0.\texttt{cp}(a_0); !;\\
&\ a_1.\texttt{set:ai}; a_0.\texttt{set:a}(a_1); a_4.\texttt{set:a}(a_1); a_2.\texttt{cp}(x_1))^{\omega}
\end{aligned}$$

which also has a finite representation.

The termination predicate and the apply operator can both be extended to regular threads using the sign instruction in the obvious way by

$$\begin{aligned} R^{\mathcal{M}}_{a_i.\mathtt{set:s}\circ T,n+1} &= \{\alpha \in \mathcal{M}^{Var} \mid \alpha[a_i := \mathbf{s}(\alpha(a_i))] \in R^{\mathcal{M}}_{T,n}\}, \text{ and} \\ (a_i.\mathtt{set:s} \circ T) \bullet \alpha &= T \bullet \alpha[a_i := \mathbf{s}(\alpha(a_i))]. \end{aligned}$$

We then have the following completeness result.

Theorem 5.8. *Let I be an instruction sequence and $k \in \mathbb{N}$ be such that $[\![I]\!]^k_{\mathcal{M}}$ is a total mapping on all signed cancellation meadows $\mathcal{M}$. Then there exists a straight-line instruction sequence J which uses at most 8 auxiliary variables such that $[\![I]\!]^k_{\mathcal{M}} = [\![J]\!]^k_{\mathcal{M}}$ for all cancellation meadows $\mathcal{M}$.*

Proof. The propositions 4.2 and 4.4 extend straightforwardly to signed cancellation meadows. Thus $|I|$ computes a term $t \in Ter(\Sigma_{ms}, \{x_0, \ldots x_k\})$. It remains to show that t can be computed by a finite and test-free thread that uses at most 8 auxiliary variables. From Corollary 5.5 it follows that t is provably equal to a sum of quotients of signed polynomials. Then, following the proof of Proposition 4.9, it suffices to prove that a signed polynomial can be computed by a finite and test-free thread using at most 6 auxiliary variables. To these ends, we invoke Lemma 5.6. Thus we may assume that there exist polynomials $t_1, t_{1_1}, \ldots, t_{1_n}, \ldots, t_i, t_{i_1}, \ldots, t_{i_n}, \ldots, t_{3n}, t_{3n_1}, \ldots, t_{3n_n}$ such that

$$t = \Sigma^{3^n}_{i=1} \Pi^n_{j=1} 0_{\phi(\mathbf{s}(t_{i_j}))} \cdot t_i$$

where $\phi(\mathbf{s}(t_{i_j})) \in \{\mathbf{s}(t_{i_j}), 1 + \mathbf{s}(t_{i_j}), 1 - \mathbf{s}(t_{i_j})\}$.

From Lemma 4.8 it follows that a polynomial t' can be computed by a finite and test-free thread using 3 auxiliary variables. Say

$$i_1 \circ \cdots \circ i_k \circ y.\mathtt{cp}(a_0) \circ \mathsf{S}$$

computes t'. Then

$$i_1 \circ \cdots \circ i_k \circ a_0.\mathtt{set:s}(a_0) \circ y.\mathtt{cp}(a_0) \circ \mathsf{S}$$

computes $\mathbf{s}(t')$ using the same variables. Thus also $\phi(\mathbf{s}(t'))$ can be computed by a finite and test-free thread using 3 auxiliary variables. Hence $0_{\phi(\mathbf{s}(t_{i_j}))} \cdot t_i$ can be computed with 4 auxiliary variables by a finite and test-free thread. Therefore it takes at most 5 auxiliary variables to compute $\Pi^n_{j=1} 0_{\phi(\mathbf{s}(t_{i_j}))} \cdot t_i$ and 6 to compute t by a finite thread without any tests. □

6 Conclusions and future work

We have described an algebraic execution system that can be used to analyze properties of instruction sequences. It is especially designed to perform calculation on the signed rational numbers. We have proven that total instruction sequences can be computed by straight-line programs with a bound supply of auxiliary variables.

Important computer algorithms based on discrete Fourier transformations can be expressed within the signed rational numbers extended with sin and π. For future work, we aim at examining equivalence and simplification problems for this kind of straight-line instruction sequences. However, it is yet unclear to us where straightening starts to fail.

References

[1] J.A. Bergstra and I. Bethke. Polarized process algebra and program equivalence. In J.C.M. Baeten, J.K. Lenstra, J. Parrow, and G.J. Woeginger, editors, *Automata, Languages and Programming, 30th International Colloquium, ICALP 2003, Eindhoven, The Netherlands, June 30 - July 4*, Springer-Verlag, LNCS 2719:1-21, 2003.

[2] J.A. Bergstra, I. Bethke, and A. Ponse. Decision problems for pushdown threads. *Acta Informatica*, 44(2):75–90, 2007.

[3] J.A. Bergstra, Y. Hirschfeld, and J.V. Tucker. Meadows and the equational specification of division. *Theoretical Computer Science*, 410(12–13):1261–1271, 2009.

[4] J.A. Bergstra and M.E. Loots. Program algebra for sequential code. *Journal of Logic and Algebraic Programming*, 51(2):125–156, 2002.

[5] J.A. Bergstra and C.A. Middelburg. Thread algebra for strategic interleaving. *Formal Aspects of Computing*, 19(4):445–474, 2007.

[6] J.A. Bergstra and A. Ponse. Combining programs and state machines. *Journal of Logic and Algebraic Programming*, 51:175–192, 2002.

[7] J.A. Bergstra and A. Ponse. *A Generic Basis Theorem for Cancellation Meadows.* arXiv:0803.3969v2, 2008.

[8] J.A. Bergstra, A. Ponse, and M.B. van der Zwaag. Tuplix Calculus. *Scientific Annals of Computer Science*, 18:35–61, 2008.

[9] J.A. Bergstra and J.V. Tucker. The rational numbers as an abstract data type. *Journal of the ACM*, 54(2), April, 2007.

[10] I. Bethke and P. Rodenburg. *The initial meadows.* arXiv:0806.2256, 2008.

[11] I. Bethke, P. Rodenburg and A. Sevenster. *The structure of finite meadows.* arXiv:0903, 2009.

[12] D.B. Bui and A.V. Mavlyanov Theory of program algebras. *Ukrainian Mathmatical Journal*, 36(6):761–764, 1984.

[13] D.B. Bui and A.V. Mavlyanov Mutual derivability of operations in program algebra. I *Cybernetics and Systems Analysis*, 24(1):35–39, 1988.

[14] D.B. Bui and A.V. Mavlyanov Mutual derivability of operations in program algebra. II *Cybernetics and Systems Analysis*, 24(6):1–6, 1988.

[15] O.A. Ibarra and S. Moran. Probabilistic algorithms for deciding equivalence of straight-line programs. *Journal of the Association for Computing Machinery*, 30(1):217–228, 1983.

[16] O.A. Ibarra and B.S. Leininger. On the simplification and equivalence problems for straight-line programs. *Journal of the Association for Computing Machinery*, 30(3):641–656, 1983.

[17] A. Ponse and M.B. van der Zwaag. An introduction to program and thread algebra. In A. Beckmann et al. (editors), *Logical Approaches to Computational Barriers: Proceedings CiE 2006*, LNCS 3988, pages 445-458, Springer-Verlag, 2006.

[18] J. von Wright. *An Interactive Metatool for Exploring Program Algebras.* Turku Centre for Computer Science, TUCS Technical Report No. 247, March, 1999.